Bearer Token Authentication

What is Bearer Token Authentication?

The Problem: API Authentication

🤔 **Question**: How do mobile apps and web APIs identify users?

Traditional web apps: Use sessions and cookies
APIs and mobile apps: Need something different!

Why sessions don't work for APIs:

- Mobile apps can't handle cookies easily
- APIs are often stateless
- Cross-domain requests are complex

What is a Bearer Token?

Simple Definition

A **bearer token** is like a **digital ticket** 🎫

- **Bearer** = "whoever holds this token"
- **Token** = a string that proves identity
- **No username/password needed** for each request

How Bearer Tokens Work

- We already discussed JWT.
- JWT is one of the token formats, and Bearer is how you send it.
    - JWT (JSON Web Token) – the most popular format.
    - Opaque tokens – random strings with no readable structure (e.g., h38djE8s9eD7w01kWqLs...).
    - Custom formats – some systems may define their token formats.

```
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxMjMsImV4cCI6MTYzMjQ4...
                ▲                    ▲
             Scheme              JWT Token
```

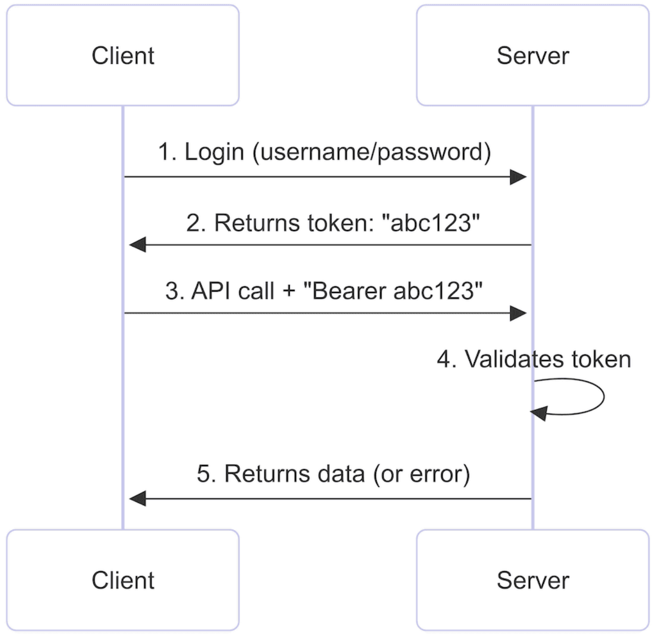
Real Example

```
Authorization: Bearer abc123xyz789
```

Think of it like:

- Concert ticket: Show it → Get in
- Bus pass: Flash it → Ride the bus
- Bearer token: Send it → Access API

Step-by-Step Process


Client
Server
1. Login (username/password)
2. Returns token: "abc123"
3. API call + "Bearer abc123"
4. Validates token
5. Returns data (or error)
Client
Server

The Flow

1. **Login once** → Get your token
2. **Keep the token** safe
3. **Send token** with every API request
4. **Server checks** if token is valid

Bearer vs Sessions vs Cookies

Method	How it works	Best for
Cookies	Browser automatically sends	Traditional websites
Sessions	Server stores user state	Web applications
Bearer	Client sends token manually	APIs, mobile apps

Key Differences

Sessions: "Server remembers you"

Bearer: "You prove who you are each time."

Bearer tokens are:

- ✅ Stateless (server doesn't store anything)
- ✅ Perfect for APIs
- ✅ Work with any client (mobile, web, etc.)

Simple Example

- bearer_auth.php
- api.php
    - accessing with curl
    - accessing with JavaScript

bearer_auth.php

- bearer_auth.php is the Bearer Token Authentication Helper.
    - It has simple functions for handling bearer token authentication

getBeareToken

- Extract bearer token from Authorization header
    - It uses regex pattern `preg_match('/Bearer\s+(.*)$/i'`

```
function getBearerToken() {
    $headers = getallheaders();

    // Check if Authorization header exists
    if (isset($headers['Authorization'])) {
        // Extract token from "Bearer TOKEN_HERE" format
        if (preg_match('/Bearer\s+(.*)$/i', $headers['Authorization'], $matches)) {
            return trim($matches[1]);
        }
    }
    return null;
}
```

isValidToken

- Simple token validation (for demo purposes)
    - In real applications, check the database for expiration

```php
function isValidToken($token) {
    // Demo tokens - in real app, check database
    $validTokens = [
        'abc123' => 'john_doe',
        'xyz789' => 'jane_smith',
        'def456' => 'admin_user',
        'student123' => 'student',
        'teacher456' => 'teacher'
    ];
    return isset($validTokens[$token]) ? $validTokens[$token] : false;
}
```

generateSecureToken

- Generate a secure random token

```
function generateSecureToken() {
    return bin2hex(random_bytes(32)); // 64 character hex string
}
```

requireAuth

- Require authentication for an endpoint
    - Call this at the start of protected endpoints

```php
function requireAuth() {
    $token = getBearerToken();
    if (!$token) {sendJsonError(401, 'Bearer token required'); }

    $user = isValidToken($token);
    if (!$user) { sendJsonError(401, 'Invalid or expired token'); }
    return $user;
}
function sendJsonError($statusCode, $message) {
    http_response_code($statusCode);
    header('Content-Type: application/json');
    echo json_encode(['error' => $message]);
    exit;
}
```

api.php - Protected API endpoint example

```php
<?php
require_once 'bearer_auth.php';

// Get the token from request
$token = getBearerToken();
if (!$token) {
    http_response_code(401);
    echo json_encode(['error' => 'Token required']);
    exit;
}
// Validate token
$user = isValidToken($token);
if (!$user) {
    http_response_code(401);
    echo json_encode(['error' => 'Invalid token']);
    exit;
}
// Success! Return protected data
echo json_encode([
    'message' => 'Welcome to protected API!',
    'user' => $user,
    'data' => ['item1', 'item2', 'item3']
]);
?>
```

Accessing api.php using cURL

- We can access the server via `api.php`.
    - We have the bearer token "student123".
- We can access the API server only with the bearer token.

```
> curl http://localhost:8000/api.php
{"error":"Token required"}

> curl -X GET "http://localhost:8000/api.php" \
     -H "Authorization: Bearer student123" \
     -H "Content-Type: application/json"
{"message":"Welcome to protected API!","user":"student","data":["item1","item2","item3"]}
```

Accessing api.php using JavaScript

```javascript
// Store token (after login)
const token = 'student123';

// Make API call with token
fetch('localhost:8000/api.php', {
    method: 'GET',
    headers: {
        'Authorization': `Bearer ${token}`,
        'Content-Type': 'application/json'
    }
})
.then(response => response.json())
.then(data => console.log(data));
```

Example

client_demo.html

Step 1: Login

HTML

- Inputs (username and password), and click the button to display a placeholder for the bearer

```html
<div class="container">
    <h2 class="step">Login to Get Token</h2>
    <div class="form-group">
        <label for="username">Username:</label>
        <input type="text" id="username" value="student" placeholder="Try: student, teacher, admin_user">
    </div>
    <div class="form-group">
        <label for="password">Password:</label>
        <input type="password" id="password" value="student123" placeholder="Password">
    </div>
    <button onclick="login()">Login</button>
    <div id="loginResponse"></div>
    <div id="tokenDisplay" class="token-display" style="display: none;">
        <strong>Your Bearer Token:</strong>
        <div id="tokenValue"></div>
    </div>
</div>
```

JavaScript

- Getting the placeholder information in HTML

```
async function login() {
    const username = document.getElementById('username').value;
    const password = document.getElementById('password').value;
    const responseDiv = document.getElementById('loginResponse');
    if (!username || !password) {
        showError(responseDiv, 'Please enter both username and password');
        return;
    }
```

- It accesses `login.php` using the POST method with username and password.

```
    try {
        const response = await fetch('login.php', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({ username, password })
        });
```

- It waits for the response from the server and displays the returned information.

```
        const data = await response.json();
        if (response.ok) {
            currentToken = data.token;
            showSuccess(responseDiv, 'Login successful!');

            // Show token
            document.getElementById('tokenDisplay').style.display = 'block';
            document.getElementById('tokenValue').textContent = currentToken;

            // Enable API button
            document.getElementById('apiButton').disabled = false;
        } else {
            showError(responseDiv, data.error || 'Login failed');
        }
    } catch (error) {
        showError(responseDiv, 'Network error: ' + error.message);
    }
}
```

Step 2: Access Protected API using the Token

HTML

```
    <div class="container">
        <h2 class="step">Access Protected API</h2>
        <p>Once you have a token, use it to access protected resources.</p>
        <button onclick="accessProtectedAPI()" id="apiButton" disabled>Access Protected API</button>
        <div id="apiResponse"></div>
    </div>
```

JavaScript

- Access protected_api.php with bearer token

```
async function accessProtectedAPI() {
    const responseDiv = document.getElementById('apiResponse');
    if (!currentToken) {
        showError(responseDiv, 'Please login first to get a token');
        return;
    }
    try {
        const response = await fetch('protected_api.php', {
            method: 'GET',
            headers: {
                'Authorization': `Bearer ${currentToken}`,
                'Content-Type': 'application/json'
            }
        });
```

- Get the information from the server and display it.

```
            const data = await response.json();
            if (response.ok) {
                showResponse(responseDiv, JSON.stringify(data, null, 2));
            } else {
                showError(responseDiv, data.error || 'API request failed');
            }
        } catch (error) {
            showError(responseDiv, 'Network error: ' + error.message);
        }
    }
```

Step 3: Manual Token Test

HTML

```html
    <div class="container">
        <h2 class="step">Manual Token Test</h2>
        <p>Try entering a token manually or test invalid tokens.</p>
        <div class="form-group">
            <label for="manualToken">Bearer Token:</label>
            <input type="text" id="manualToken" placeholder="abc123, xyz789, or def456">
        </div>
        <button onclick="testManualToken()">Test Token</button>
        <div id="manualResponse"></div>
    </div>
```

JavaScript

- Using the given token, we try to access the API.

```
async function testManualToken() {
    const token = document.getElementById('manualToken').value;
    const responseDiv = document.getElementById('manualResponse');
    if (!token) {
        showError(responseDiv, 'Please enter a token');
        return;
    }
    try {
        const response = await fetch('protected_api.php', {
            method: 'GET',
            headers: {
                'Authorization': `Bearer ${token}`, 'Content-Type': 'application/json'
            }
        });
        const data = await response.json();
        if (response.ok) { showResponse(responseDiv, JSON.stringify(data, null, 2)); } else {
            showError(responseDiv, data.error || 'Token validation failed');
        }
    } catch (error) {
        showError(responseDiv, 'Network error: ' + error.message);
    }
}
```

login.php

Step 1: Get JSON input

```php
$input = json_decode(file_get_contents('php://input'), true);
```

Step 2: Retrieve username and password

```php
if (!isset($input['username']) || !isset($input['password'])) {
    sendJsonError(400, 'Username and password required');
}
$username = $input['username'];
$password = $input['password'];
```

Step 3: Check the users' database

```php
$users = [ ... ] // DB in an array
// Validate credentials
if (!isset($users[$username]) || $users[$username] !== $password) {
    sendJsonError(401, 'Invalid username or password');
}
```

Step 4: Generate token, store in DB, and return JSON

```
$demoTokens = [ ... ]
$token = $demoTokens[$username];
$demoToken[...] = $token;

// Return success with token
sendJsonSuccess([
    'message' => 'Login successful',
    'token' => $token,
    'user' => $username,
    'expires_in' => 3600 // 1 hour (demo value)
]);
```

protected_api.php

Step 1: Get a bearer token to check authentication

```php
// Require authentication - this will exit if no valid token
$user = requireAuth();
```

Step 2: Return protected data

- We can add user-specific data

```php
$protectedData = [
    'message' => 'Welcome to the protected API!',
    'authenticated_user' => $user,
    'data' => [
        'secret_info' => 'This is confidential data',
        'server_info' => 'PHP ' . phpversion()
    ]
];
// Add user-specific data
if ($user === 'admin_user') {
    $protectedData['admin_data'] = [
        'admin_tools' => ['user_management', 'system_logs']
    ];
}
// Return the protected data
sendJsonSuccess($protectedData);
?>
```

index.php

- This script has all the test code for the interactive demo.

test_curl.sh

- We can download `test_curl.sh` from the index.php menu.

```
<div class="demo-card">
    <h3>💻 Command Line</h3>
    <p>Test with cURL commands and see the raw HTTP requests and responses.</p>
    <a href="test_curl.sh" download>Download Script →</a>
</div>
```

Run test_curl.sh

```
> bash test_curl.sh
bash test_curl.sh
🔐 Bearer Token Authentication Examples
=======================================

Step 1: Log in to get a bearer token
======================================

Log in with valid credentials:
curl -X POST http://localhost:8000/login.php \
     -H "Content-Type: application/json" \
     -d '{"username":"student","password":"student123"}'

Try this command:
{"message":"Login successful","token":"student123","user":"student","expires_in":3600}

Step 2: Use the token to access the protected API
==============================================

Access protected endpoint with valid token:
curl -H "Authorization: Bearer student123" \
     http://localhost:8000/protected_api.php

Try this command:
{"message":"Welcome to the protected API!","authenticated_user":"student",
"timestamp":"2025-08-06 22:55:21",
"data":{"secret_info":"This is confidential data",
"user_permissions":["read","write"],
"server_info":"PHP 8.4.11"},
"student_data":{"enrolled_courses":["ASE230"],"grades":["A","B+","A-"],"next_assignment":"Bearer Token Project"}}
```

```
Step 3: Test with an invalid token
==================================

Try with an invalid token:
curl -H "Authorization: Bearer invalid_token" \
     http://localhost:8000/protected_api.php

This should return an error:
{"error":"Invalid or expired token"}

Step 4: Test without a token
=============================

Try without any token:
curl http://localhost:8000/protected_api.php

This should also return an error:
{"error":"Bearer token required"}

Summary:
========
✅ Valid token: Returns protected data
❌ Invalid token: Returns 401 error
❌ No token: Returns 401 error

Valid tokens for testing:
- student123 (user: student)
- teacher456 (user: teacher)
- abc123 (user: john_doe)
- xyz789 (user: jane_smith)
- def456 (user: admin_user)

Other users you can log in with:
- username: teacher, password: teacher456
- username: admin_user, password: admin789
- username: john_doe, password: password123
- username: jane_smith, password: secret456
```

Token Management

Generating Secure Tokens

- There are many ways to generate secure tokens.

```php
<?php
function generateSecureToken() {
    // Generate cryptographically secure random token
    return bin2hex(random_bytes(32)); // 64 character hex string
}

function createTokenForUser($userId) {
    $token = generateSecureToken();
    $expiry = time() + (60 * 60); // 1 hour from now
    // Store in database
    // INSERT INTO tokens (token, user_id, expires_at) VALUES (?, ?, ?)
    return $token;
}
?>
```

Error Handling

Proper HTTP Status Codes

```php
<?php
function sendUnauthorized($message = 'Unauthorized') {
    http_response_code(401);
    header('Content-Type: application/json');
    echo json_encode(['error' => $message]);
    exit;
}
function sendForbidden($message = 'Forbidden') {
    http_response_code(403);
    header('Content-Type: application/json');
    echo json_encode(['error' => $message]);
    exit;
}
// Usage
if (!$token) { sendUnauthorized('Bearer token required'); }
if (!isValidToken($token)) { sendUnauthorized('Invalid or expired token');}
?>
```

Key Takeaways

Bearer Token Authentication Enables

- 🔐 **Stateless authentication** for APIs
- 📱 **Mobile app** authentication
- 🌐 **Cross-domain** API access
- ⚡ **Scalable** authentication systems

Remember

1. **Bearer tokens** = digital tickets for API access
2. **Always use HTTPS** for security
3. **Tokens should expire** for safety
4. **Perfect for APIs** and mobile apps
5. **Simpler than sessions** for stateless applications

Where Bearer Tokens Are Used

1. **Mobile Apps** 📱
   - Instagram, Twitter, Facebook apps
   - Banking applications
2. **Single Page Applications** 💻
   - React, Vue, Angular apps
   - Modern web dashboards

3. **API Integrations** 🔗
    - Payment processing (Stripe, PayPal)
    - Cloud services (AWS, Google Cloud)
4. **Microservices** ⚙️
    - Service-to-service communication
    - Distributed applications